University of Rennes 1

Master 2 recherche en informatique - ISTIC/IRISA

Integrating Diagnostic and Repair to Ensure the Quality of a Composition of Web Services

Author:
Muhammad Ali Memon

Supervisors:
Marie Odile Cordier
Sophie Robin
Laurence Rozé

June 1, 2011

Abstract in French L'informatique orientée service repose sur la composition dynamique de web services pour répondre à la demande d'un utilisateur. Un défi important conditionnant une utilisation réelle des web services consiste à surveiller leur exécution et à les rendre capables de réagir à des dysfonctionnements imprévus. Ceci peut se faire en utilisant les mécanismes de traitement d'exceptions. Mais ceux-ci ne permettent de réagir que de manière prédéfinie et à des problèmes locaux et prévus dès la conception des services. Pourtant, dans des environnements dynamiques tels qu'Internet, les web services peuvent être sujets à des dysfonctionnements imprévus. De plus, la gestion locale des fautes par traitement d'exceptions ne tient pas compte des interactions entre services, ce qui en limite l'efficacité. En cas par exemple de pannes se propageant à travers les services avant d'être détectées, l'important est de localiser la panne à la source du dysfonctionnement et de réparer le service correspondant. Dans ce contexte, le travail de stage consisté à étudier une gestion distribuée mais coordonnée des mécanismes de réparation. La difficulté vient du fait que les réparations sont exécutées localement, mais qu'une stratégie globale doit être assurée pour tenir compte des interactions entre les différents services. L'objectif est de proposer une architecture de diagnostic-réparation permettant cette fonctionnalité et d'en détailler les mécanismes.
Mots cles : QoS, Surveillance Web services, Diagnostic et Réparation.

Abstract in English Service-Oriented Computing is based on dynamic composition of web services to meet the demand of a user. A major challenge in conditioning actual use of web services is to monitor their performance and enable them to react to unexpected malfunctioning. This can be done using the mechanisms of exception handling. But they do react in a predefined manner and local issues have to be planned at the services design time. However, in dynamic environments like the Internet, web services may be subject to unexpected malfunctioning which may not be handled with repair mechanisms defined at design time . In addition, local management ignores errors during the interactions between services, which limit their effectiveness. Such failures may also propagate through the services before being detected, and the key is to find the problem at the source of the malfunction and repair the service. In this context, this work is dedicated to study a distributed but coordinated and dynamic management of repair mechanisms. The difficulty is that repairs are carried out locally, but a global approach must be ensured to take into account interactions between different services. Our objective is to propose a diagnostic-repair architecture and mechanisms for this feature in detail .
Keywords: QoS, Web Services Monitoring, Diagnostic and Repair.

Contents

1 Introduction **4**

2 Web Services **6**
2.1 Composition of Services 6
2.2 XML 7
2.3 SOAP 7
2.4 WSDL 8
2.5 UDDI 8
2.6 WS-BPEL 8

3 Context **9**
3.1 Diagnostic from WS-Diamond Project 9
3.2 The Diabolo Project 10
3.2.1 Motivating Example/ Case Study 10
3.2.2 Faults and Symptoms 11
3.2.3 Architecture 12
3.3 Taser 18

4 My Contribution **18**
4.1 Modified Diabolo Architecture 19
4.2 Scenarios 21
4.3 Implementation 23
4.3.1 Domain Theory Rules with Prolog 24
4.3.2 Taser reloaded 24
4.3.3 Integration in progress 24

5 Related Work/ State of the art **25**

6 Conclusion **28**

7 Appendix **28**
7.1 Fault Scenarios 29
7.2 Taser Snapshots 36

1 Introduction

The emerging paradigm of Service Oriented Computing consists of allowing the composition of distributed services, thus providing value-added services. In an environment such as Internet, web services are invoked dynamically to provide B2B solutions. Web services can be subject to unexpected failures for which recovery strategies were not planned at design time. New requirements are posed by the need for detection mechanisms and recovery strategies, managing and handling unexpected exceptions. In general, it is difficult and costly to plan all possible recovery strategies at design time. Moreover, this kind of local fault management does not span across individual services and thus limits the effectiveness of recovery strategies. The difficult cases are those, where a fault propagates through services without being detected. The fault cannot be handled in a correct way without locating the primary cause and identifying the faulty services. It becomes evident in the system in terms of a failure that may arise from different error states and may be caused by different faults in system components. For instance, let us consider an e-commerce context, suppose a client orders a product via an e-shop. If the shop service has an in exact product code in its database, the process goes on through the partner services until the problem is detected. To make application more flexible, faults must be detected as soon as possible and properly handled.

Failure is caused by faults and faults originate in operations associated with activities in web services in composition or individually. Faults may comprise multiple failing operations. When a fault is permanent, we assume that all further executions of the faulty operations may result in faulty behavior. A fault is temporary, or transient, when subsequent executions of this operation are correct.

Figure 1 presents two processes that may result in cancellation of order on the shop due to not satisfying reservation list: (a) a data acquisition error, ordering "eggs" and "teak" instead of "eggs and tea", and (b) a stock error happening on the warehouse. Here, we notice that two distinct errors that happen on two distinct services can result in the same local symptom, hence the necessity of diagnosing the system globally in order to repair it in an adequate way. In the case of the data acquisition error, only an interaction with the after-sales service can correct the mistake, but in the second case, asking an alternate warehouse for the missing item may prevent the customer from receiving an incomplete parcel.

The basic common strategies proposed in literature is establishing a closed control loop based on MAPE Loop/model [8]: Monitoring, Analysis, Planning and Execution. Two ways to realize MAPE loops are global composers

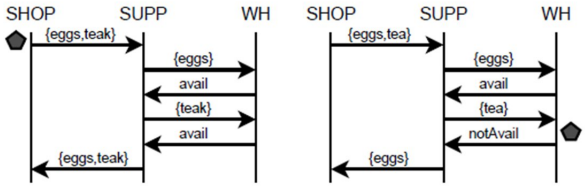

Figure 1: a)*Data acquisition error* b)*Stock error* [10]

and local adapters. In case of any misbehavior of a service; an approach based on global composers substitutes a faulty service with the correct one and by re-composition. While in an approach based on local adapters, each service has the ability to adapt itself in a local way. Global composers get global consistency but are usually computationally expensive. Local adapters are cheaper and efficient in taking local decisions, but adaptations that are locally acquired may not maintain a global view. Solving adaptation locally might not benefit in the attempt of building a global and consistent adaptation for the application.

An hybrid approach has been proposed by Marie Odile Cordier, Sophie Robin and Roberto Micalizio named "Diabolo", which establishes trade off between high flexibility of local adapters and consistency of global composers. Diabolo don't allow to change the initial composition; but empower each activity with the ability of selecting the best way to get its local goal among the number of alternative options/modalities; these alternative execution modalities make each activity very flexible and apt to solve part of the adaptation problem locally. However, an activity can locally select the best execution modality by only having a global view of the composition. For this reason a roadmap is attached to the user's request which is shared and easily accessible by each activity within the orchestration. This roadmap contains information about orchestration and hence helps activities to adapt themselves locally. Activities are also allowed to modify some information only relevant to that particular activity in order to maintain this roadmap updated for request's execution status. A MANAGER module of this loop will be in charge of (re)initializing the roadmap, while another module ADAPTER keeps it up to date.
The goal of this work is firstly to propose how activities can be enriched with alternative execution modalities. Secondly, if time permits realizing to implement a prototype of the Diabolo approach with a case study that has already been proposed by Diabolo.

The rest of the report is organized as follows. Section 2 briefly introduces

web services and its terminologies. Section 3 presents the context of this work the work. Section 4 details my contribution during this internship. Section 5 is dedicated to related approaches and their comparison with the approach. Section 6 concludes and gives perspectives. Section 7 is dedicated to an appendix for detailing scenarios and GUI snapshots.

2 Web Services

For many years scientists are trying to develop frameworks and architectures that can make available applications for use no matter which language/tools/platforms were used to develop them. Service-Oriented Architecture (SOA) and web services have become quite popular in this regard. When all the platforms have access to the web using internet web browsers, different platforms could interact with each other. For this purpose, platforms to work together, web services/applications are developed. "Web services are defined as self-contained, modular units of application logic which provide business functionality to other applications via an Internet connection. Web services support the interaction of business partners and their processes by providing a stateless model of atomic synchronous or asynchronous message ex-changes. These atomic message exchanges can be composed into longer business interactions by providing message exchange protocols, that show the mutually visible message exchange behavior of each of the partners involved" [13]. Web services communicate using open protocols like SOAP(section 2.3) and can be described using UDDI(section 2.5). The basic web services platform is XML (section 2.2) plus HTTP. Web services have two types of advantages; reusable application-components and connecting existing software. Web services have three basic platform elements SOAP, WSDL and UDDI. SOAP is an XML based protocol to let applications exchange information over HTTP. SOAP is designed to communicate via internet and is also platform/language independent. WSDL is an XML-based language for locating and describing Web services. UDDI is a directory service where companies can register and search for Web services. [2]

2.1 Composition of Services

Composition of web services has a high potential in business and enterprise application integration. The two approaches used for composition of web services: Industrial approach and the Semantic web approach. The Industry treats web service as a standardized interface to business processes and express it in WSDL. Four types of communication are defined involving a ser-

vice's operation: one-way, notification, request-response and solicit-response. However Semantic web approach treats web service as an additional piece of information (semantic annotations). Semantic annotations are assertions and their properties are expressed in Resource Description Language (RDF). WSDL provides a function-centric description of web services covering inputs, outputs, and exception handling. The Semantic Web provides a process level description of the service, while Industry uses BPEL4WS approach for the composition of web services proposed by (IBM and MICROSOFT). Semantic approach translates DAML-L/RDF notations in Prolog. Both compositions allow the customization of the plan execution at runtime i.e., to select a particular branch of execution or to loop until an exit condition is satisfied. Authors in [13] concludes that none of the approaches has developed a true planning solution to the service composition problem so far.

2.2 XML

XML stands for extensible Markup Language. It was designed to transport and store data[1]. XML is a markup language like HTML. It is now being widely used as the most common tool for data transmission between all sorts of applications over internet. It helps to keep the important data separate, this helps concentrate on layout and display. XML data is stored in plain text format making it independent from software and hardware. This makes it easier to create data that can be shared by different applications. It simplifies platform changes. It helps define our own tags and our own document structure. A lot of internet languages are created from XML like XHTML,WSDL, WAP and WML, RSS, RDF and OWL, and SMIL .

2.3 SOAP

It is important to allow internet communication between programs for application development through remote procedure calls (RPC). But RPC raises compatibility and security problem; firewalls and proxy servers will normally block this kind of traffic. A better way to communicate is over HTTP, because HTTP is supported by all browsers and servers. SOAP was created to replace RPC and communicate over HTTP without any hindrance. SOAP (Simple Object Access Protocol) is an XML based protocol used for exchanging messages between applications over HTTP. SOAP communicates via internet and runs on different operating systems with different technologies and language [1].

2.4 WSDL

Web Services Description Language (WSDL) is an XML based language for describing web services and how to access them. WSDL is a document that specifies the location of services and operators (or methods), that the service exposes. If a client wishes to use an operation on a web service, he can read WSDL description of operations offered by the service and the description of XML Schema data types used by these operations.

2.5 UDDI

UDDI (Universal Description Discovery and Integration) is an XML based registry on the internet to register and locate web services. Businesses worldwide, in order to list them on internet, register their services in this registry in order to be located by other services. UDDI is designed to be interrogated by SOAP messages and provides access to WSDL describing the protocol bindings and message formats required to interact with the web services listed in its directory.

2.6 WS-BPEL

The Web Services Business Process Execution Language (WS-BPEL) provides a language for describing business processes and business communication protocols. WS-BPEL was designed, aiming to extend the interaction of web services to support business transactions. WS-BPEL 2.0 is an OASIS Standard since april 2007. It describes model and grammar to define business process behavior based on interactions between process and its partners. The WS-BPEL process defines how multiple service interactions with partners are coordinated to achieve a business goal, as well as the state and the logic necessary for this coordination. WS-BPEL can separate both public and private or internal aspects of the business process and supports both. This protects business partners from the need to reveal all their internal decision making and data management to one another. WS-BEPL can be used for both executable and abstract processes, where executable process describes the actual behavior of web services in business interactions and abstract process represent publicly observable behaviors. WS-BPEL leverages other web services standards such as SOAP and WSDL for communication and interface description. By describing the inbound and outbound process interfaces in WSDL, BPEL enables them to be easily integrated into other processes or applications [3].

3 Context

In this section, I shall try to build a focused view of work targeted for this internship. The title of this internship is "Integration of Diagnostic and Repair to ensure the QoS for composition of web services". However Diagnostic module [10, 9, 7] has already been developed by DREAM during the s(WS-Diamond) project(presented in section 3.1). This internship is based on two parts. First to build an adapter module and second is to incorporate that adapter module within the Diabolo architecture to construct complete self healing framework for web applications (integration of architecture depends on internship time constraint). To achieve this goal a preliminary theoretical study has already conducted done by Marie Odile Cordier and Sophie Robin in collaboration with Roberto Micalizio from University of Torino named Diabolo. It includes an initial representation of aself healing architecture and a motivating example. A web service simulating tool has also been developed by DREAM team called Taser explained in section 3.3. Initially Taser was not included within this work but was considered later, when our architecture was being implemented. We then decided to modify Taser to simulate our architecture. In this section, first I start by briefly introducing the diagnostic module and then, I present Diabolo project with the premier architecture and then, I describe Taser.

3.1 Diagnostic from WS-Diamond Project

In this section, I present the diagnostic task accomplished by the Dream team under the WS-Diamond project summarizing four papers [10, 9, 7]. WS-Diamond Project was funded by the EU commission under the FET (Future and Emerging Technologies) from 2005 to 2008. This project aimed at the development of a framework for self-healing web services, that will be able to self-monitor and self-diagnose the causes of a failure, and to self-recover from functional failures and from non-functional failures. Diagnostic task mainly concerns diagnosis of data semantic faults, i.e. faults on input data, or database contents, or even human faults such as bad typing or wrong handling. Most of the time, faults propagate from one service to another. Moreover, two distinct faults that happen on two distinct services can result in the same observable symptoms. Approach adopted for diagnosing faults is a chronicle recognition approach, currently in use for monitoring complex industrial systems [6]. This approach is extended to the specific context of web services for distributed systems. In this regard two platforms Matrac[10] and Carde-Crs[10, 9] have been designed . Matrac is dedicated to web services, whose choreography model is described in WS-CDL. For Carde-Crs,

interaction model is not a priori known. Comparison of both platforms is given in [7].

3.2 The Diabolo Project

Orchestrations of services provoke global failures, if affected by different types of faults. To overcome this issue many approaches have a specific orchestrator service that is responsible for (re)adaptation of the whole orchestration, therefore maintaining QoS by solving a (re)composition. In the diabolo approach adaptation is not treated as a re composition activity but rather it is proposed to keep orchestration static and encapsulate it into MANAGER module which manages adaptation loop. A customer's request is received by MANAGER, which prepares a road map for the request and then attaches that road map to the request and then the request is processed by workflow as usual. Road map contains some pieces of information, that enables each activity to choose the best treatment in case of alternatives. When request reaches at the end of execution of workflow smoothly, MANAGER stops the process and terminates all service instances in normal model.

If any misbehavior is detected and notified by local monitor, MANAGER suspends the process and finds the best way to treat the request with a new abnormal possible roadmap. MANAGER is helped by two other modules called ANALYSER/DIAGNOSER and ADAPTER. They both together suggest adaptation plan to the MANAGER. MANAGER keeps alive same instances of services and tries to reuse both data and results achieved by those services as far as possible in order to reuse them in the next attempt of processing the same request.

MANAGER keeps track of different attempts and errors encountered during previous attempts for the request in order to determine unrecoverable services. For example MANAGER can decide no more to submit the request, when either it cannot fulfilled or some unrecoverable hardware or network failure occurs.

3.2.1 Motivating Example/ Case Study

In this section, I present an illustrative example from the Diabolo. This example concerns e-commerce, and more exactly electronic documents composition. A customer accesses an internet site through its welcome page (WELcome) to order an electronic pdf document (see a possible input screen for the WELcome service in Figure 2. The document will be composed of a sequence of images (one image on one page) and the customer specifies a keyword for each image (for instance "Mont-Saint-Michel" or "Georges Clooney"

to get any image representing the Mont-Saint-Michel city or the well-known actor). To answer the customer's query, an orchestration of services is composed of the WELcome service, that calls the images-SUPplier service, in charge of finding one image for each keyword, and then asks the customer for confirmation after having displayed the keywords for which images can be provided. In fact, the image-SUPplier may be unable to match some keywords with corresponding images, in that case the customer decides whether the album should be composed the same or not. After the confirmation by the customer, the customer's email address is sent by the WELcome and the images by the images-SUPplier service to the COmposer service. This last service is in charge of composing the final pdf document and emailing it to the customer, who then confirms (or not) the good reception of the pdf file. That terminates the request process by triggering the end of waiting processes for all the services of the orchestration.

3.2.2 Faults and Symptoms

Faults can occur during the request process by any service and can propagate through other services. In the example above, we consider four different faults. The observations we rely on to detect the faults are called symptoms, more often symptoms observed in one service that is not the one which is responsible for the fault. We consider two kinds of symptoms. The first type is the non correspondence between expected behavior and observed behavior of the service. In our case customer can be unhappy with the pdf file, (s)he received and pushed the Grumpy Button instead of Happy Button. The second type represents abnormal delays (communication in services and timeouts). Symptoms are depicted as stars in Figure 3. In this example we restrict to consider four faults, one by the WELcome service, two by the SUPplier and one by the COmposer.

Email-Fault: It corresponds to an error in the email address, that means the user provides an email address that is not the one (s)he is currently using.

Image-Database-Fault: It corresponds to an inconsistency between a keyword and the image stored in the database (the "Rennes" keyword is linked to an image representing "Paris").

Corrupted-Image-Fault: This fault occurs when one or more images retrieved from SUpplier's database and sent to the COmposer are corrupted and prevent the COmposer from performing composition.

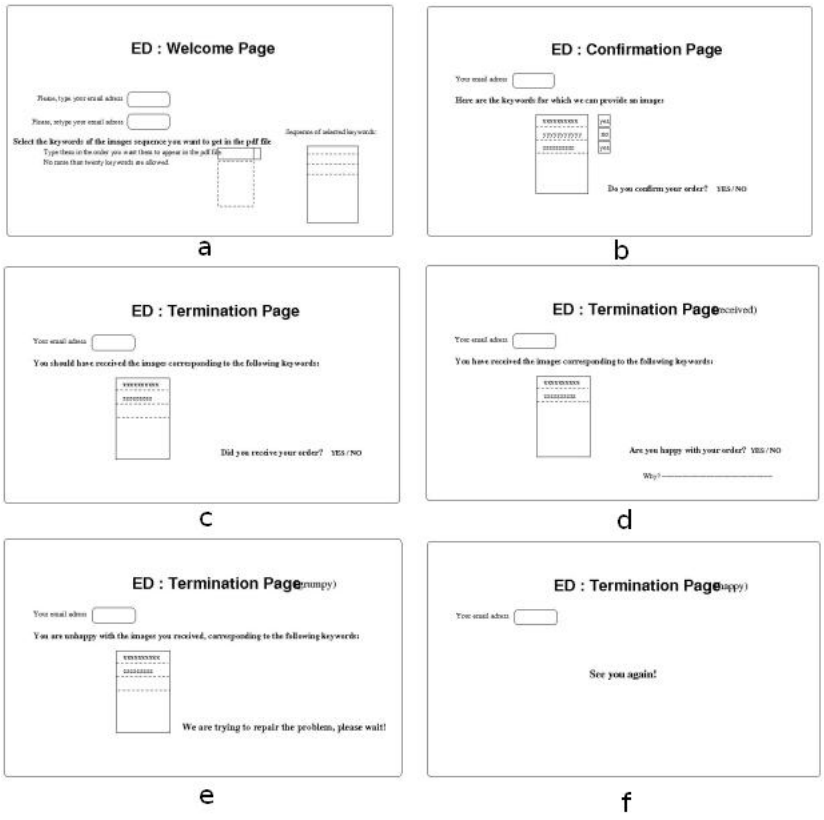

Figure 2: *Different screenshots of a simple GUI for the WELcome service. In a) the customer fills in a form with her/his email and the list of keywords. In b) the WELcome service shows the customer which images can be actually provided by the SUPplier. In c) the WELcome checks the reception of the pdf file. In d) and e) the customer provides the WELcome with positive and negative feedback, respectively*

Sending-Composer-Fault: This fault occurs when the COmposer for some technical reason(network etc) fails to send the pdf file to the customer.

3.2.3 Architecture

Figure 4 shows the architecture of our approach. It comprises of four successive modules corresponding to the MAPE loop. The role of each module starts from low level which is called EXE module.

THE EXE module corresponds to a workflow that comprises of distributed services, where each service itself composed of activities, control structures, sequences loops, input and output variables etc. The EXE module is in charge of processing a request fulfilling QoS constraints. As explained

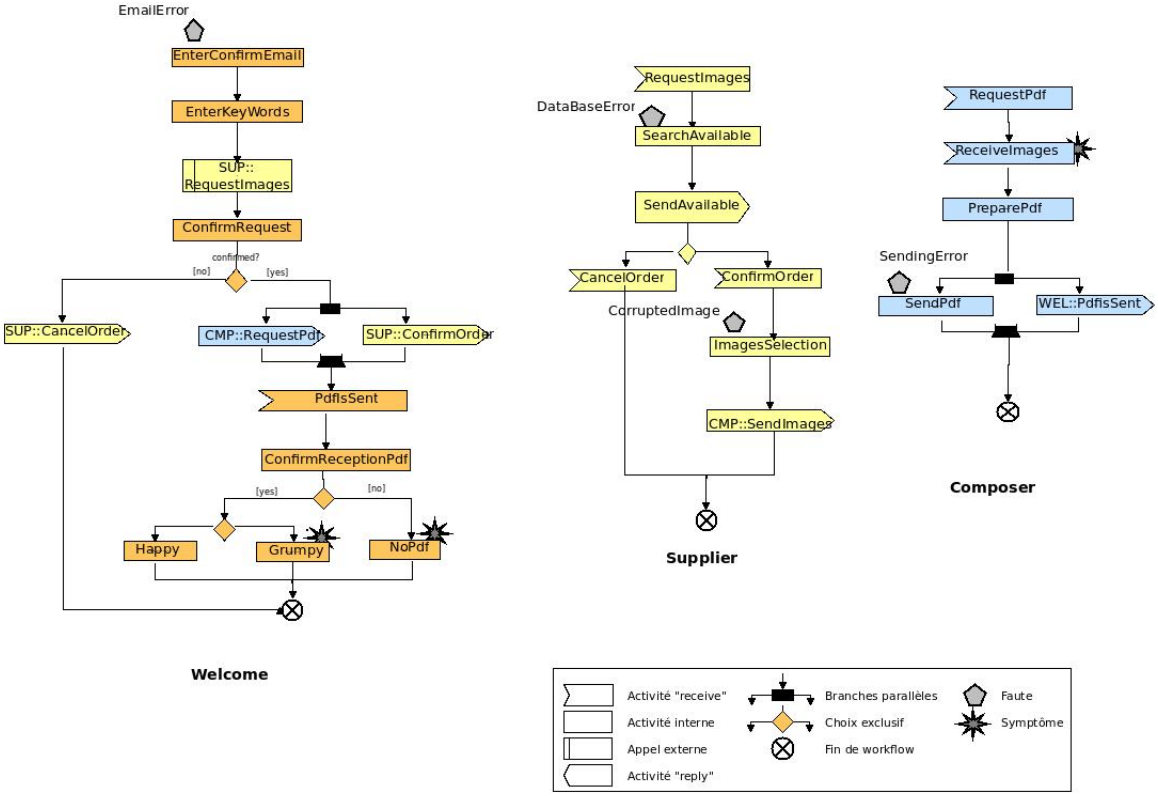

Figure 3: *Services, Activities and Interactions*

earlier in introduction activities are enriched with a set of alternatives. When execution calls an activity. It chooses the best alternative according to its context and the road map which is attached to th request.

The MONITOR module is in charge of monitoring every service locally. It logs the actual behavior of a service and detects any misbehavior. If something goes wrong (recognizing a symptom), It determines, whether it is worth suspending the current process. If yes, then it stills the execution and enters in a diagnosis/local adaptation/re-execution loop. In this case, it sends monitored information(local diagnosis) to ANALYSER.

The ANALYSER module computes a global diagnosis given local diagnosis communicated from MONITOR module and look for primary causes of problems. ANALYSER is awaked by one of the monitors. It may start the discussion with monitors to build a global view of what happened, that resulted in misbehavior. It looks for the primary cause and computes a list of correctly achieved goals and transmits them to the ADAPTER.

The ADAPTER module is in charge for the best way to process the

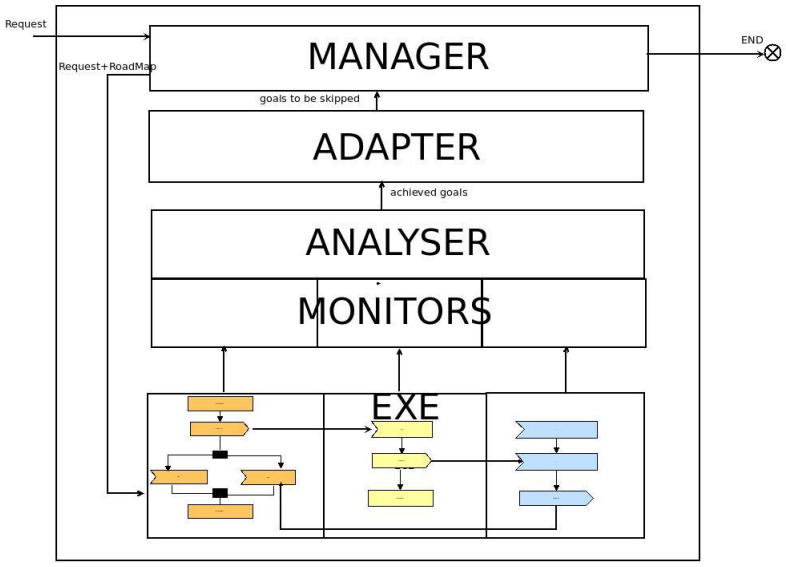

Figure 4: *Architecture of Diabolo*

request. It is responsible for adapting the original plan to a new road map, determining which subgoals have to be achieved and which subgoals may be skipped as there is no need to recompute them because they have already been achieved correctly and their results are stored and reusable in the next run. ADAPTER uses the composition model, built by WS-COMPOSER, that represents an abstract goal oriented view of the global interaction of the orchestration.

The MANAGER module is in charge of the whole control loop. It means addressing the user's request and deciding to stop the process, either normally or abnormally (in case if any problem detected and all the adaptation carried out resulted unsuccessful and there is not enough time or no more any repair alternative). It is also in charge of preparing a road map(considering all the Qos constraints) that contains the global context and staples it to the request. The road map then travels with the request and passes on from all the services involved in workflow of the process.

GD-Graph: WS-COMPOSER module is not part of our architecture and is not built when orchestration is made, but it is created when the processing of the request starts. It gives some useful piece of information in the form of an abstract goal-oriented view, which groups a plan of goals. It is used by ANALAYSER and ADAPTER to decide which goals are correctly achieved and which goals still remain to be achieved. Figure 5 illustrates the

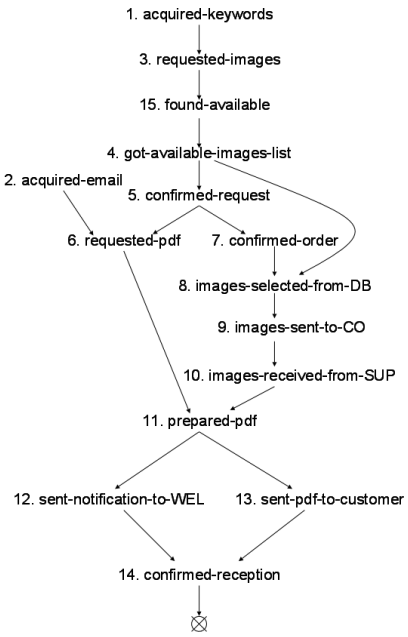

Figure 5: *GD-Graph for the Album Composer Scenario*

GD-Graph for our Album Composer Scenario. GD-Graph illustrates dependencies between activities belonging to different services, whereas it hides the notion of services. The main purpose of GD-Graph is to capture transversal dependencies among activities through which failures may propagate.

In order to make activities adaptable, they are enriched with alternatives. Each activity is attached with goal(s), as both ADAPTER and the MANAGER have an abstract goal-oriented view of the services composition. Table 6 reports the matching between activities their goals. A goal is achieved by means of a number of activities, particularly when we consider activities which synchronize two services; for instance activity CMP::Requestpdf in WELcome and activity Requestpdf in COmposer are abstracted by a single goal requested-pdf, which is satisfied when a specific signal has been sent from the WELcome to the COmposer, and a precise Composer's activity has been started. For example in our Album Composer example, activity *ImageSelection* have two execution modalities. HighQualityDBSelection and LowQualityDBSelection. By default, this activity chooses LowQualityDBSelection database for image selection. For instance, in first execution, one or more images selected from LowQualityDBSelection were corrupted, that caused system failure. In second execution the activity will be choosed HighDBSelection to reprocess the request.

Service	Activity	Goal
WELcome	EnterConfirmEmail	acquired-email
WELcome	EnterKeywords	acquired-keywords
SUPplier	RequestImages	requested-images
SUPplier	SearchAvailable	found-available
SUPplier	SendAvailable	got-available-images-list
WELcome	ConfirmRequest	confirmed-request
COmposer	RequestPdf	requested-pdf
SUPplier	ConfirmOrder	confirmed-order
SUPplier	ImageSelection	images-selected-from-DB
SUPplier	SendImages	images-sent-to-CO
COmposer	ReceiveImages	images-received-from-SUP
COmposer	PreparedPdf	prepared-pdf
COmposer	SendPdf	sent-pdf-to-customer
WELcome	PdfisSent	sent-notification-to-WEL
WELcome	ConfirmReceptionPdf	confirmed-reception

Figure 6: *Matching Activities with Goals*

Each goal in the GD graph is associated with a boolean flag called avail(g) that is true for a goal that can be stored. When a goal achieved during the execution of the system, and locally stored by the activity, its results can be re-used during a subsequent re-execution of the services. Otherwise flag avail(g) is false. Table 7 reports goals availability for our example. In principle, goals that aim to produce or exchange data have the avail flag set to true; for example goals *acquired-keywords* and *acquired-email* are achieved when, by means of some activities, the customer's list of keywords and his/her email are stored within the local memory of the service. Therefore data associated with these goals can be recycled. On the contrary *confirm-order* and *sent-notification-to-WEL* don't produce any data. They represent only synchronization points, so their flag remain false. The flag avail just states that a goal can be reused, but it does not specify, under which conditions the goal is actually reusable. The actual reusability of a goal depends also on the status of the stored goal. An erroneous goal cannot be reused even if its avail flag is true. For this reason the ADAPTER complements the GD-Graph with a Domain Theory(DT).

Domain Theory Rules: DT specifies under which circumstances a goal has not to be achieved. Intuitively, an activity can be skipped either, when its goal is already available, or when there is not any necessity to achieve the

Goal	Number	Avail
acquired-keywords	1	true
acquired-email	2	true
requested-images	3	false
found-available	15	true
got-available-images-list	4	true
confirmed-request	5	true
requested-pdf	6	false
confirmed-order	7	false
images-selected-from-DB	8	true
images-sent-to-CO	9	false
images-received-from-SUP	10	false
prepared-pdf	11	true
sent-notification-to-WEL	12	false
sent-pdf-to-customer	13	false
confirmed-reception	14	false

Figure 7: *Goals Availability for the Album Composer Scenario*

activity's goal. The first type of DT that we use, has the following template:

$$\text{achieved(g)} \wedge \text{avail(g)} \text{ —-> to-be-skipped(g)}$$

If goal_a has been correctly already achieved during previous execution and is still available, then there is no need to recompute goal_a again. For example achieved(acquired-email) $\wedge$ avail(acquired-email) —-> to-be-skipped(g). During the re-execution phase, the activity responsible for such goal EnterConfirmEmail, will retrieve the email from its local memory, that was acquired from the customer previously. The second type of the rule is:

$$[\forall \text{ g (parent(G,g)} \wedge \text{ to-be-skipped(g)] -> to-be-skipped(G)}$$

where all the children of a goal G may be skipped, if the goal G itself may be skipped. For example if *prepared-pdf* has already been achieved, now there is no need to get *requested-pdf* nor to get *received-images-from-SUP* again.

In conclusion LOOP model proposed by the Diabolo approach matches the MAPE model. However not all four modules reside on the same level. In fact MONITOR and EXECUTION phases are performed locally. On the contrary Diagnoser and ADAPTER phases are carried out at global level. This architecture tries to make a good balance between the cheapness and

flexibility of local adapter approach and global consistency property of global composers approach.

3.3 Taser

Taser is a tool for simulating web services Workflow. Taser has been developed by Ronan Quitin during his Master's internship with DREAM team. Taser was developed with Java Swing for (graphical user Interface) GUI, and Java-RMI for communication inter-simulator. Taser can read files directly from services written in XML. The simulation can be done in two different modes: automatic and interactive. Interactive mode allows the user to choose which activity it will run. When an activity is simulated, a log line is generated in a file. Automatic mode simply selects an executable activity at random and runs it. The communication part is simulated directly by using multiple instances of simulator(illustrated as a snapshot at the end in appendix 7.2). each instance simulates a service, and RMI technology is used. Thus when simulator reaches an Invoke activity, it will make a remote call to another instance. An instance, that receive remote call, executes Receive corresponding to that of Invoke. Taser is distributed in nature, if we want to simulate composition of n services. We need to load n instances of Taser in order to perform remote procedure calls. Class Diagram of Taser is given in Figure 8.
Use of Java RMI demands knowledge of remote IP positions. A registry is created similar to that of UDDI, whose mission is to list all existing services and provide services to IP services listed by name. Thus in the simulation, if the simulator needs to run Invoke activity from one service, it is enough for it to query the UDDI to get the IP of the service it wishes to make its remote call. The dynamism is preserved despite the layer Java RMI technology.

Taser has two interfaces, graphic and console. Console mode was not correctly functional initially, but I corrected it during my internship. However, the graphic mode is functional and has advantage for use in demonstration. Indeed, during a simulation, GUI allows to see graphically the commonly simulated workflow and especially to see the execution path taken.

4 My Contribution

Objective of this internship is to propose an adapter module that define, how activities can be enriched with alternative execution modalities . Secondly, if time permits, then realizing Diabolo approach with the adapter module with a case study that has already been proposed by Diabolo. In this regard

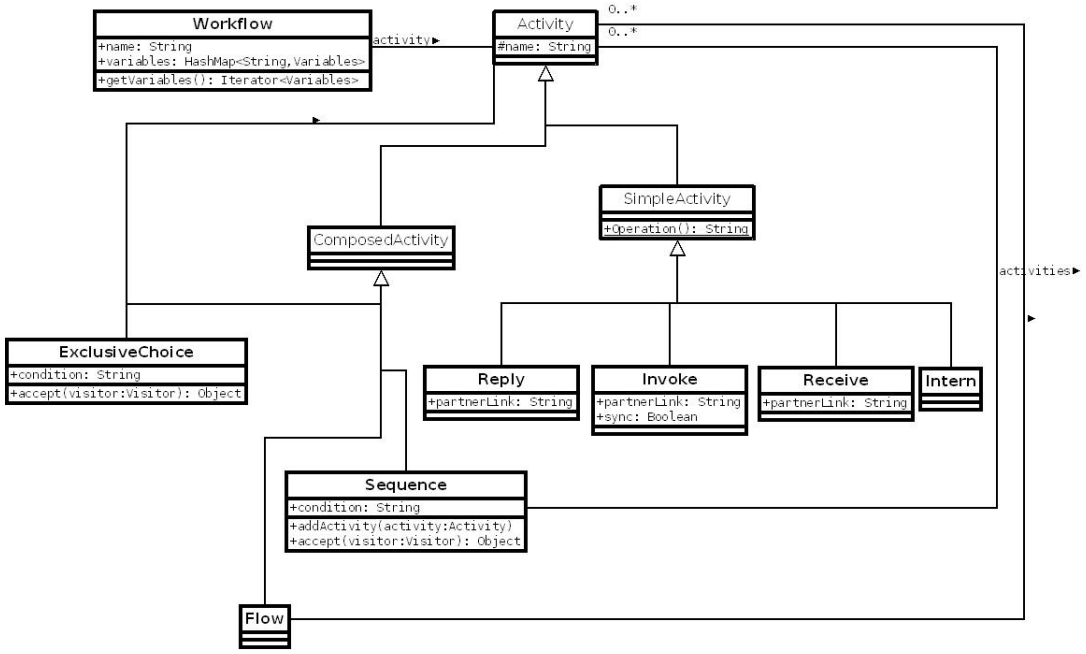

Figure 8: *Initial class diagram of Taser representing all activities of web service to build workflow*

at the time of writing this report first phase of enriching activities has been proposed and successfully implemented. Currently, I am working on incorporating my adapter module within Diabolo approach to yield the complete self healing framework. In this section I shall explain my contribution during this internship. In section 4.1, I detail modifications for Diabolo architecture. In section 4.2, I present scenarios/case studies, that I built in order to clearly identify the functioning of each module of the architecture. In section 4.3, I list the implementation details of our approach.

4.1 Modified Diabolo Architecture

In this section, I detail my proposition for each module in the architecture.

EXE: As explained earlier this module is in charge for executing the workflow for processing the request. I decided to use Taser, a tool for simulating web services workflow (explained in section 3.3). I modified initial architecture of Taser in order to make it compatible with the architecture of Diabolo. I needed to enrich activities with different alternatives named execution modalities to make them adaptable. A web service has four basic activities; *Intern*, *Invoke*, *Receive* and *Reply*. I proposed a new activity

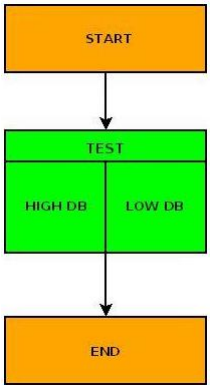

Figure 9: *Represents a web service that has one start and one end activity and in between one meta activity, which encapsulate two intern activities HIGHDB and LOWDB*

named *"Meta"*. Meta will reside on the top of the intern activities. Meta activity will encapsulate two or more alternative Intern activities, while processing the request, Meta activity decides which Intern activity to execute considering QoS constraints that passed through road map with the request. In case of a failure caused by one Intern activity from the alternatives of Meta activity, another alternative activity will be choses in the next run for a new road map, if MANAGER decides to re-execute the request. Image shown in figure 9 represents a simple service that contain one start and one end Intern activities and in between one Meta activity with two alternatives of Intern activities. One alternative searches images from database of high quality (HIGHDB) and another searches images from database of low quality (LOWDB).

MONITOR: This module looks for trouble while execution in EXE module; it detects any misbehavior in the form of any symptom or system failure. If system fails at some stage of execution, it finds out the localization of failure while executing an activity(goal). It asks ANALYSER to diagnose it. This module is not our concern as it has already been developed during WS-Diamond Project. However in this work, at the moment of writing this report, I am implementing a simulation of it in order to connect it with GD-graph to get localization of failure of both activity and goal.

ANALYSER: When it receives symptom of misbehavior as input from MONITOR module, it performs diagnosis to find out activity (goal) that caused system failure or any misbehavior. This module is a part of diagnostic task (not the context of my work) and has already been done, but will be integrated afterwards in Diabolo architecture.

ADAPTER: It receives as input the list of achieved goals from ANAL-

YSER module and generate a list of goals to be skipped in the next run. ADAPTER is responsible for generating the possible adaptation of original plan to a new plan for re-processing. ADAPTER module uses domain theory rules (explained previously in section 3.2.3), GD-Graph and avail flags (for the goals) to generate the list of goals to be skipped for new abnormal plan.

MANAGER: As explained before the MANAGER is in charge of controlling the MAPE loop and take decisions to stop the process or re-process the request with the new road map. Initially road map was not clearly defined, that how it will be used for taking decisions. I proposed some parameters like number of executions, time out and user priority to incorporate in it. The decision of re-processing is taken at run time by MANAGER. Currently I am working on integrating all these modules together to simulate whole architecture.

4.2 Scenarios

I analysed four scenarios EMAIL-FAULT, SENDING-COMPOSER-FAULT, GRUMPY and CORRUPTED-IMAGE corresponding to each fault Email-Fault, Sending-Composer-Fault Image-Database-Fault and Corrupted-Image-Fault respectively. In each scenario, I consider single fault and detail input and output for each module that, and show step by step how fault is handled dynamically by our architecture. In this section, I present only the scenario corresponding to the CORRUPTED-IMAGE fault and the remaining three scenarios are presented in the appendix at the end of this report.

CORRUPT-IMAGE

In this scenario COmposer service fails to compose the pdf file, because one or more than one images are corrupted. ANALYSER proposes one diagnosis, corresponding to the corrupted-image-fault.

MONITOR:
The goal of this module is to identify a problem in the activity, particulary achieved by recognizing a symptom. There may be more than one symptom corresponding to one fault or one symptom corresponding to more than one fault. In this scenario, there is only one symptom.

Output from MONITOR

Symptom: Composer could not perform composition successfully.
Detecttion of a problem: A problem occurred while achieving the "prepared-pdf" goal at "Prepared-pdf " activity.

ANALYSER:
The goal of this module is to diagnose the activity (goal) from where the fault originated. It generates the list of achieved goals.

Input from MONITOR

Symptom: at (11)prepared-pdf goal.

Output from ANALYSER

Fault: corrupted-image-fault.
Erroneous Goal Origin: (8)images-selected-from-DB

Correctly achieved Goals: (1)acquired-keywords, (2)acquired-email, (3)request-images, (15)found-available, (4)got-available-images-list, (5)confirmed-request, (6)requested-pdf, (7)confirmed-order.

ADAPTER
This module gets a list of correctly achieved goals from ANALYSER. It applies domain theory rules to generate a list of goals that can be skipped in the next run for adaptation and send that list to MANAGER.

Input from ANALYSER

Achieved Goals: (1)acquired-keywords, (2)acquired-email, (3)request-images, (15)found-available, (4)got-available-images-list, (5)confirmed-request, (6)requested-pdf, (7)confirmed-order.

Application of Domain Theory(DT) Rules

First Rule achieved(g) $\wedge$ avail(g) —-> to-be-skipped(g)

Using the facts: achieved(*acquired-keywords*), achieved(*acquired-email*), achieved(*request-images*), achieved(*got-available-images-list*), achieved(*confirmed-request*), achieved(*requested-pdf*), achieved(*confirmed-order*) and following instances of the first type rule:

(1.1) achieved(*acquired-keywords*) $\wedge$avail(*acquired-keywords*) —-> to-be-skipped(*acquired-keywords*)

(1.2) achieved(*acquired-email*) ∧avail(*acquired-email*) —-> to-be-skipped(*acquired-email*)
(1.15) achieved(*found-available*) ∧avail(*found-available*) —-> to-be-skipped(*found-available*)
(1.4) achieved(*got-available-images-list*) ∧avail(*got-available-images-list*) —-> to-be-skipped(*got-available-images-list*)
(1.5) achieved(*confirmed-request*) ∧ avail(*confirmed-request*) —-> to-be-skipped(*confirmed-request*)

I deduce to-be-skipped(*acquired-keywords*), to-be-skipped(*acquired-email*), to-be-skipped(*found-available*), to-be-skipped(*got-available-images-list*), and to-be-skipped(*confirmed-request*)

Second Rule [∀ g (parent(G,g) ∧ to-be-skipped(g)] -> to-be-skipped(G)]

Using the second rule, I deduce to-be-skipped(*request-images*)

Output from ADAPTER

Goals-to-be-Skipped: (1)acquired-keywords, (2)acquired-email, (3)request-images, (15)found-available, (4)got-available-images-list, (5)confirmed-request.

MANAGER
In this scenario, Manager decides to re-process the request with a new Road map consisting of the list of goals to be skipped and an additional piece of information denoting that, this is the second execution for the same request.
EXE
In the second execution, activity responsible for retrieving images will choose images from another database of HighQuality.
activity-id : SUP::ImagesSelection
activity goals : retrieved-DB-images
set-of-modalities : MediumQualityDBSelection, HighQualityDBSelection
policies:
if not the first execution of the request : HighQualityDBSelection
by default : MediumQualityDBSelection

4.3 Implementation

I divided the implementation of the architecture into three steps.

1. Implement Domain Theory rules for ADAPTER model in Prolog.

2. Modify Taser to add Meta activity and its execution modalities.

3. Integrate all the modules to achieve final functionality.

I have successfully implemented first and second steps and now I am working on the third step. I detail the implementation of first and second phase respectively.

4.3.1 Domain Theory Rules with Prolog

I have implemented Domain Theory Rules with Prolog, as it is the best choice for logic programming. I used SWI-prolog (a comprehensive free software for licensed under the Lesser GNU Public License).This program has the unchanged rules for GD-Graph (to specify dependency) and avail flag information for all the goals. It takes the list of achieved goals (input for ADAPTER from ANALYSER) as input and generate list of goals to be skipped. Finding a solution for interface between Prolog and Java took some time. I use jpl (JAVA to Prolog Interface) for interface that also comes with the same bundle of Prolog that includes SWI-Prolog compiler.

4.3.2 Taser reloaded

Taser is a simulator for service workflow. Taser have two modes: graphical and console. Graphical mode was completely functional, while console mode was not functional. I have modified it and corrected the console mode. Snapshot for console mode is given at the end in appendix section 7.2. Initially Taser did not support the execution modalities in activities, but I modified its initial architecture and added a new class named *Meta*. Taser has two abstract classes "Activity" and "SimpleActivity" that extends "Activity". Four classes "Intern", "Invoke", "Receive" and "Reply" extends "SimpleActivity" class representing four basic activities of service. I added new "Meta" class that also extends "SimpleActivity" class and to encapsulate execution modalities I added a private collection of "Intern" activities in Meta. Modified Architecture of Taser is shown in Figure 10.

4.3.3 Integration in progress

Currently I am implementing integration for all the modules of project Diabolo. EXE module now consists of modified Taser with Meta activity. Taser loads the three services of Album Composer scenario and start simulating these services and processing the request. During execution, if everything goes fine, it ends the execution and quits. If any misbehavior occurs, which is detected by MONITOR, it suspends the execution and informs ANALYSER about the symptom and local(goal) at which system encountered mis-

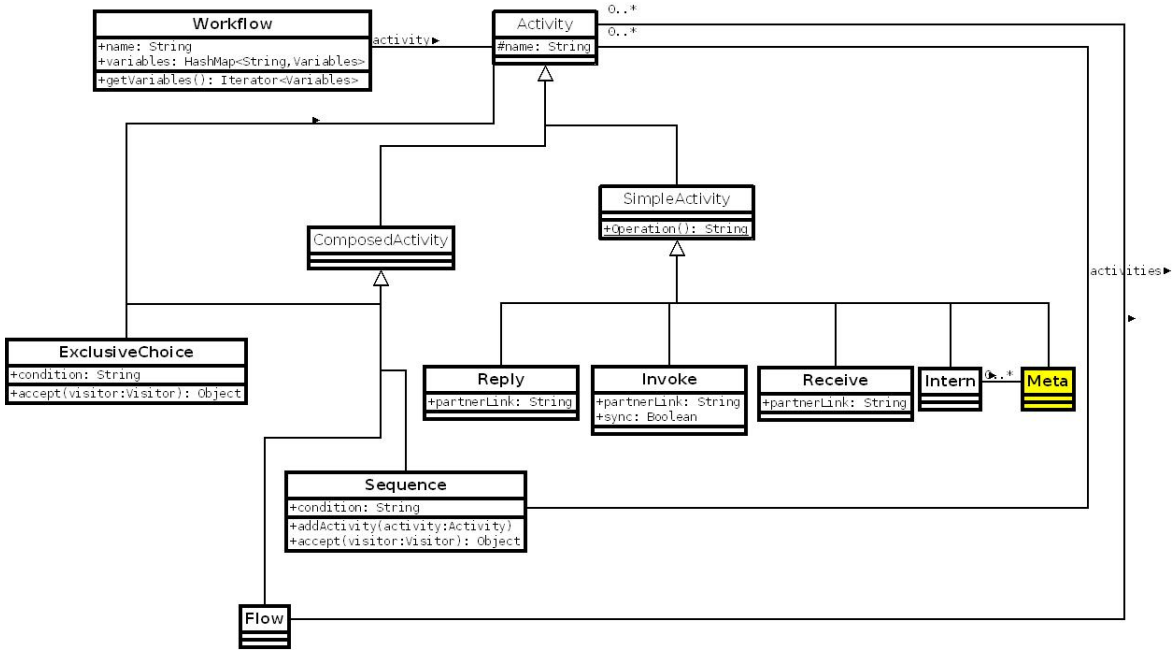

Figure 10: *Modified class Diagram of Taser with added Meta activity*

behavior. ANALYSER then computes the origin of failure (erroneous goal (activity)) with diagnosis and compute correctly achieved goals and transmit them to the ADAPTER. ADAPTER then computes to-be-skipped goals with domain theory rules and transmits them to the MANAGER. Now it is MANAGER, which then decides to reprocess it or not.

5 Related Work/ State of the art

Service-Oriented Architectures enforce a strong separation of concerns through the loose coupling of the services, well suited to make distributed applications. However, those distributed applications can be subject to variations in the quality of service they provide. It can be due to various factors, such as the volatility of execution nodes that arise from failures, maintenance actions or voluntary connections or disconnections, new services may be deployed, existing ones removed, varying useres load and varying user's demands that require different levels of quality of service. All these factors lead to the necessity for dynamic adaptation [11].

Several approaches have been proposed for recovering web services from failures in composition of web services.

In ([5], 2007), Onyeka Ezenwoye and S. Masoud Sadjadi present an approach called generic proxy. Generic proxy checks the invocation of services and tries to match these invocations with the specified policies. If it does not find any policy it follows the default (common) behavior. If a policy exits, it follows the specific behavior. The default behavior of the proxy is to consult the registry (UDDI) to substitute faulty service with an appropriate service that implements the same interface. For specific behavior it takes one of the actions according to the policy. It invokes the recommended service mentioned in the policy or finds and invokes another service for the monitored service or retries the invocation of monitored service. The policies are associated with monitored services and are specified in a configuration file that is loaded at startup in to the generic proxy. Finding appropriate service at run time depends on syntactical functional description (as represented by WSDL), which has limitations to automatic discovery and require some human interaction. Limitation of this approach is that it is adapting system on the basis of local diagnosis(local repairing) and system has to perform re-composition and start the process from beginning. It does not store the tasks that has already been performed and computed. If proxy service itself fails the whole repairing (adaptation) fails. If generic proxy is unable to find a suitable service for replacement it requires human intervention.

S.Sattanathan et al ([14], 2008) modified current ActiveBPEL engine to enhance it for self healing purpose in . For this reason they create a self healing policy called sh-policy. Sh-policy have 4 parts: Plan part details pre and post conditions per BPEL activity; Monitor part details the BPEL activities to track during performance, Diagnose part details the unexpected failures and their root causes. Recover part details solutions to recover from failures. Plan and Monitor parts are identified during BPEL process compilation, while diagnose and recover parts are identified during BPEL process execution. These four parts are associated with four modules. Plan module identifies the pre and post conditions per BPEL activity; in addition it identifies the BPEL activities to monitor at run time that have direct impact on a web service output. Monitor tracks the BPEL activities and catches the unexpected failure and records the failure details and expected type of solution in the diagnose part of sh-policy. Diagnosis module concerns the database (containing information about failures) generated from the exception list of a BPEL engine and suggest solutions to the recover part of sh-policy. Recovery module applies the solution available in the recover part of sh-policy. However it does not monitor and diagnose all the activities in the system rather it chooses specific activities depending on which activities have direct impact on a web service output. Here diagnosis is based on local basis. No

global context is considered in case of a failure originated from one service and diagnosed in another, especially in an activity which is not being monitored.

In ([12], 2007), Barbara Pernici et al present an approach for learning repair strategies for web services. When a fault is notified, the system estimates its type (permanent, transient and intermittent) with the highest probability score. Bayesian classification assigns a type to each fault(transient, permanent, intermittent) depending on previous faults and model parameters. Then the system evaluates fault context and parameters. Fault context is defined as a set of features, these features are service (identifier, provider and invoked operations etc). System then performs comparison between notified fault and previously classified faults. If comparison is positive then repair strategy of most similar fault is retrieved and applied. If repair is successful then the notified fault is added to model, otherwise several repair strategies are applied from next matching faults. Retry can be performed several times till the maximum number already fixed. For substitution a wrapper is generated in order to adapt and translate invocation. If substitution cannot be applied, a manual repair is requested. This approach does not express how a fault is notified from diagnosis. Another limitation is that when system is fixed then this approach claims that system continues the execution after that. However, it does not focus on which activities need to be re-executed from which fault propagated and caused misbehavior.

Anis Charfi et al in ([4], 2009) propose an approach for designing plug-ins. In this approach, plug-ins are designed by developers for a well defined objective. Plug-ins have two kinds of extension points: implicit and explicit. Implicit extension point is associated with the process activity, where plug-in can execute adapted logic, while explicit extension point may contain links to the web services that are provided for replacement in case if adaptation is needed. Plug-ins can be deployed and extracted at run time making self adaption dynamic. The difference between this and our approach is that this approach takes into account erroneous situations and events that are detected at an activity, but in fact this activity is not responsible for the fault, rather it is a propagating fault originated while executing a previous activity. Secondly, self adaptation at the detected location may not resolve the problem. Further self adaptation at run time for the erroneous event does not promise preserving correct results performed by previous activities and resuming the execution from where execution was stopped, which is a major priority in our approach.

6 Conclusion

The objective of this internship is to ensure the quality of service of the request while processing it even if faults occur, not only by local exception handling, but taking into account the global view given by diagnostic. Initially we had the faults (having caused the failure) computed by the diagnoser. Our idea was to see the repair plan as adaptation of the normal process and not to re-compose the services. In that context, we needed to check which service or activity has to be adopted and in which way to ensure at best the QoS without recomposing the services. Finally validating the quality of system after execution of repair plan. Our approach is based on MAPE loop. It comprises of four modules; MONITOR for detecting misbehavior, ANALYSER for identifying original location of fault, ADAPTER for proposing new adaptation strategy and finally MANAGER for controlling loop and taking decisions. MANAGER attaches a road map with each request in order to process it with the best QoS. Activities are enriched with alternative execution modalities and they chose one of them dynamically based on best QoS. I proposed a new activity for web service called "Meta" which encapsulates these execution modalities. In case of failure MANAGER decides to re-process the request with a new abnormal plan or stops execution, when there is no solution possible without re-composition. Our approach is limited to the number and extent of activities are enriched. In case in a workflow of services, if a fault occurs on an activity, which is not enriched, then no adaptation strategy can be generated for that. Eventually, our framework will stop the execution and re-composition remains the only solution (which we don't consider in this work). We use a motivating example of Album Composer Scenario for realizing the whole framework. In this work we are simulating MONITOR and ANALYSER module, but in future we can connect actual MONITOR and ANALYSER module from diagnostic task accomplished within WS-Diamond project. Another advancement is that we can replace Taser with actual BPEL engine to verify our architecture in real time environment. In addition to that, it is also required to study the links between diagnostic and repair, in order to best adapt to the diagnosis repair capabilities.

7 Appendix

Remaing three scenarios and Taser GUI snapshots are presented in this appendix.

7.1 Fault Scenarios

GRUMPY

User pushes the Grumpy button, meaning that user received the pdf file but some of the images do not correspond to his/her input keywords. ANALYSER proposes one diagnosis, corresponding to the images-supplier-database-fault denoting an inconsistency between a key word and the corresponding image stored in the supplier database.

MONITOR:
The goal of this module is to identify a problem in the activity, particulary achieved by recognizing a symptom. There may be more than one symptom corresponding to one fault or one symptom corresponding to more than one fault In this scenario, there is only one symptom corresponding to one fault.

Output from MONITOR

Symptom: User pushed the grumpy button.
Detecttion of the problem: A problem occurred while achieving the "confirmed-reception" goal attached to "ConfirmedReceptionPdf " activity from service WELcome.

ANALYSER:
The goal of this module is to find out the activity (goal) from where fault originated. It generates the list of correctly achieved goals.

Input from MONITOR

Symptom: At (14)confirmed-reception.
Fault: image-supplier-database-fault.

Output from ANALYSER

Erroneous Goal Origin: (4)got-available-images-list

Corrctly achieved Goals: (1)acquired-keywords, (2)acquired-email, (3)request-images.

ADAPTER
This module gets the list of correctly achieved goals and from ANALYSER. It then applies domain theory rules to generate list of goals that can be skipped in next run for adaptation and send that list to MANAGER.

Input from ANALYSER

Correctly achieved Goals: (1)acquired-keywords, (2)acquired-email, (3)request-images.

Application of Domain Theory(DT) Rules

First Rule achieved(g) $\wedge$ avail(g) —-> to-be-skipped(g)

Using the facts: achieved(*acquired-keywords*), achieved(*acquired-email*), achieved(*request-images*), and following instances of the first type rule:

(1.1) achieved(*acquired-keywords*) $\wedge$avail(*acquired-keywords*) —-> to-be-skipped(*acquired-keywords*)
(1.2) achieved(*acquired-email*) $\wedge$avail(*acquired-email*) —-> to-be-skipped(*acquired-email*)

I deduce to-be-skipped(*acquired-keywords*), to-be-skipped(*acquired-email*).

Second Rule [$\forall$ g (parent(G,g) $\wedge$ to-be-skipped(g)] -> to-be-skipped(G)]

No instances of this rule can be applied in this scenario.

Output from ADAPTER

Goals-to-be-Skipped: (1)acquired-keywords, (2)acquired-email.

MANAGER
In this scenario, Manager decides to re-process the request with a new Road map consisting of the list of to-be-skipped goals and an additional piece of information denoting that is the second execution for the same request.

EMAIL-FAULT

Customer pushes the time-out button, meaning that (s)he does not receive the file in the expected delay. There are two possibilities. First one is that user did not provide the correct email address. Second one is that some network fault occurred while sending file to the customer. Therefore ANALYSER proposes two diagnoses, email-fault and sending-composer-fault. ANALYSER considers these both faults and prepare list of correctly achieved goals for both faults. Similarly Adapter generates separate lists of to-be-skipped goals related to each fault and send two separate adaptation strategy to the Manager. Manager then decides new road map based on some parameters and two adaptation strategy. This scenario considers only email-fault.

MONITOR:
The goal of this module is to identify a problem in the activity, particulary achieved by recognizing a symptom. There may be more than one symptom corresponding to one fault or one symptom corresponding to more than one fault. In this scenario, there is only one symptom for more than one fault (Email fault and sending composer fault). Here I explain consider Email fault.

Output from MONITOR

Symptom: User pushes the Time Out button.
Detecttion of the problem: : A problem occurred while achieving the "confirmed-reception" goal at "ConfirmedReceptionPdf " activity from service WELcome.

ANALYSER:
The goal of this module is to find out the activity (goal) from where fault originated. It generates the list of correctly achieved goals.

Input from MONITOR

Symptom: At (14)Confirmed-reception goal.
Fault: Email-fault in the Welcome service .

Output from ANALYSER

Erroneous Goal Origin: (2)acquired-email

Correctly Achieved Goals: (1)acquired-keywords, (3)request-images, (15)found-available,(4)got-available-images-list, (5)confirmed-request, (6)requested-pdf, (7)confirmed-order, (8)images-selected-from-DB,(9)images-sent-to-CO, (10)images-received-from-SUP,(11)prepared-pdf.

ADAPTER
This module gets list of achieved goals and from ANALYSER. It then applies domain theory rules to generate list of correctly goals that can be skipped in next run for adaptation and send that list to MANAGER.

Input from ANALYSER

Correctly achieved Goals (1)acquired-keywords, (3)request-images, (15)found-available,(4)got-available-images-list, (5)confirmed-request, (6)requested-pdf, (7)confirmed-order, (8)images-selected-from-DB,(9)images-sent-to-CO, (10)images-received-from-SUP,(11)prepared-pdf.

Application of Domain Theory(DT) Rules

First Rule achieved(g) $\wedge$ avail(g) —> to-be-skipped(g)

Using the facts: achieved(*acquired-keywords*), achieved(*found-available*), achieved(*got-available-images-list*), achieved(*confirmed-request*), achieved(*images-selected-from-DB*), achieved(*prepared-pdf*) and following instances of the first type rule:

(1.1) achieved(*acquired-keywords*) $\wedge$avail(*acquired-keywords*) —-> to-be-skipped(*acquired-keywords*)
(1.15) achieved(*found-available*) $\wedge$avail(*found-available*) —-> to-be-skipped(*found-available*)
(1.4) achieved(*got-available-images-list*) $\wedge$avail(*got-available-images-list*) —-> to-be-skipped(*got-available-images-list*)
(1.5) achieved(*confirmed-request*) $\wedge$ avail(*confirmed-request*) —-> to-be-skipped(*confirmed-request*)
(1.8) achieved(*images-selected-from-DB*) $\wedge$avail(*images-selected-from-DB*) —-> to-be-skipped(*images-selected-from-DB*)
(1.11) achieved(*prepared-pdf*) $\wedge$avail(*prepared-pdf*) —> to-be-skipped(*prepared-pdf*)

I deduce to-be-skipped(*acquired-keywords*), to-be-skipped(*found-available*),

to-be-skipped(*got-available-images-list*), to-be-skipped(*confirmed-request*), to-be-skipped(*images-selected-from-DB*) and to-be-skipped(*prepared-pdf*)

Second Rule [$\forall$ g (parent(G,g) $\wedge$ to-be-skipped(g)] -> to-be-skipped(G)]

Using the second rule, I deduce to-be-skipped(*request-images*), to-be-skipped(*requested-pdf*), to-be-skipped(*confirmed-order*), to-be-skipped(*images-sent-to-CO*) and to-be-skipped(*images-received-from-SUP*)

Output from ADAPTER

Goals-to-be-Skipped: (1)acquired-keywords, (3)request-images, (15)found-available, (4)got-available-images-list, (5)confirmed-request, (6) requested-pdf, (7)confirmed-order, (8)images-selected-from-DB, (9)images-sent-to-CO,(10)images-received-from-SUP, (11)prepared-pdf.

MANAGER
In this scenario, Manager decides to re-process the request with a new Road map consisting of the list of to-be-skipped goals and an additional piece of information denoting that is the second execution for the same request.

SENDING COMPOSER-FAULT

Customer pushes the time-out button, meaning that (s)he does not receive the file in the expected delay. There are two possibilities. First one is that user did not provide the correct email address. Second one is that some network fault occurred while sending file to the customer. Therefore ANALYSER proposes two diagnoses, email-fault and sending-composer-fault. ANALYSER considers these both faults and prepare list of correctly achieved goals for both faults. Similarly Adapter generates separate lists of to-be-skipped goals related to each fault and send two separate adaptation strategy to the Manager. Manager then decides new road map/Context based on some parameters and two adaptation strategy. This scenario considers sending-composer-fault.

MONITOR:
The goal of this module is to identify a problem in the activity, particulary achieved by recognizing a symptom. There may be more than one symptom corresponding to one fault or one symptom corresponding to more than one fault. In this scenario, there is only one symptom for more than one fault (Email fault and sending composer fault). Here I explain consider composer fault.

Output from MONITOR

Symptom: User pushes the Time Out button.
Detecttion of the problem: : A problem occurred while achieving the "confirmed-reception" goal at "ConfirmedReceptionPdf " activity from service WELcome.

ANALYSER:
The goal of this module is to find out the activity (goal) from where fault originated. It generates the list of correctly achieved goals.

Input from MONITOR

Symptom: At (14)Confirmed-reception goal.
Fault: Sending-Composer-Fault in the Welcome service.

Output from ANALYSER

Erroneous Goal Origin: (13)sent-pdf-to-customer

Correctly achieved Goals: (1)acquired-keywords,(2)acquired-email, (3)request-images, (15)found-available,(4)got-available-images-list, (5)confirmed-request, (6)requested-pdf, (7)confirmed-order, (8)images-selected-from-DB,(9)images-sent-to-CO, (10)images-received-from-SUP,(11)prepared-pdf.

ADAPTER
This module gets list of correctly achieved goals from ANALYSER. It then applies domain theory rules to generate list of goals that can be skipped in next run for adaptation and send that list to MANAGER.

Input from ANALYSER

Correctly achieved Goals: (1)acquired-keywords,(2)acquired-email, (3)request-images, (15)found-available,(4)got-available-images-list, (5)confirmed-request, (6)requested-pdf, (7)confirmed-order, (8)images-selected-from-DB,(9)images-sent-to-CO, (10)images-received-from-SUP,(11)prepared-pdf.

Application of Domain Theory(DT) Rules

First Rule achieved(g) $\wedge$ avail(g) —> to-be-skipped(g)

Using the facts: achieved(*acquired-keywords*), achieved(*found-available*), achieved(*got-available-images-list*), achieved(*confirmed-request*), achieved(*images-selected-from-DB*), achieved(*prepared-pdf*) and following instances of the first type rule:

(1.1) achieved(*acquired-keywords*) ∧avail(*acquired-keywords*) —-> to-be-skipped(*acquired-keywords*)
(1.2) achieved(*acquired-email*) ∧avail(*acquired-email*) —-> to-be-skipped(*acquired-email*)
(1.15) achieved(*found-available*) ∧avail(*found-available*) —-> to-be-skipped(*found-available*)
(1.4) achieved(*got-available-images-list*) ∧avail(*got-available-images-list*) —-> to-be-skipped(*got-available-images-list*)
(1.5) achieved(*confirmed-request*) ∧ avail(*confirmed-request*) —-> to-be-skipped(*confirmed-request*)
(1.8) achieved(*images-selected-from-DB*) ∧avail(*images-selected-from-DB*) —-> to-be-skipped(*images-selected-from-DB*)
(1.11) achieved(*prepared-pdf*) ∧avail(*prepared-pdf*) —-> to-be-skipped(*prepared-pdf*)

I deduce to-be-skipped(*acquired-keywords*),to-be-skipped(*acquired-email*), to-be-skipped(*found-available*), to-be-skipped(*got-available-images-list*), to-be-skipped(*confirmed-request*), to-be-skipped(*images-selected-from-DB*) and to-be-skipped(*prepared-pdf*)

Second Rule [∀ g (parent(G,g) ∧ to-be-skipped(g)] -> to-be-skipped(G)]

Using the second rule, I deduce to-be-skipped(*request-images*), to-be-skipped(*requested-pdf*), to-be-skipped(*confirmed-order*), to-be-skipped(*images-sent-to-CO*) and to-be-skipped(*images-received-from-SUP*)

Output from ADAPTER

Goals-to-be-Skipped: (1)acquired-keywords, (2)acquired-email, (3)request-images, (15)found-available, (4)got-available-images-list, (5)confirmed-request, (6) requested-pdf, (7)confirmed-order, (8)images-selected-from-DB, (9)images-sent-to-CO,(10)images-received-from-SUP, (11)prepared-pdf.

MANAGER
In this scenario, Manager decides to re-process the request with a new Road map consisting of the list of to-be-skipped goals and an additional piece of information denoting that is the second execution for the same request.

Input from ADAPTER

Goals-to-be-Skipped: (1)acquired-keywords, (2)acquired-email, (3)request-images, (15)found-available, (4)got-available-images-list, (5)confirmed-request, (6) requested-pdf, (7)confirmed-order, (8)images-selected-from-DB, (9)images-sent-to-CO,(10)images-received-from-SUP, (11)prepared-pdf.

7.2 Taser Snapshots

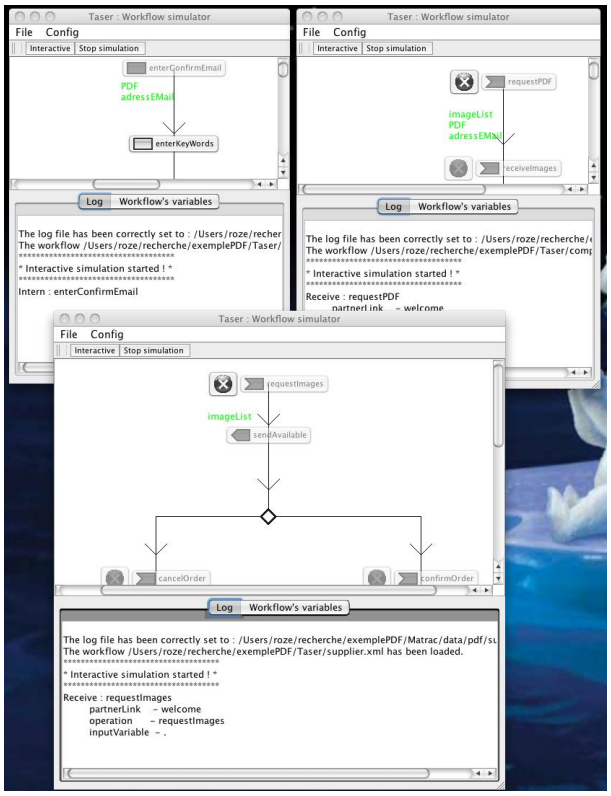

Figure 11: *Snapshot of Taser graphical mode*

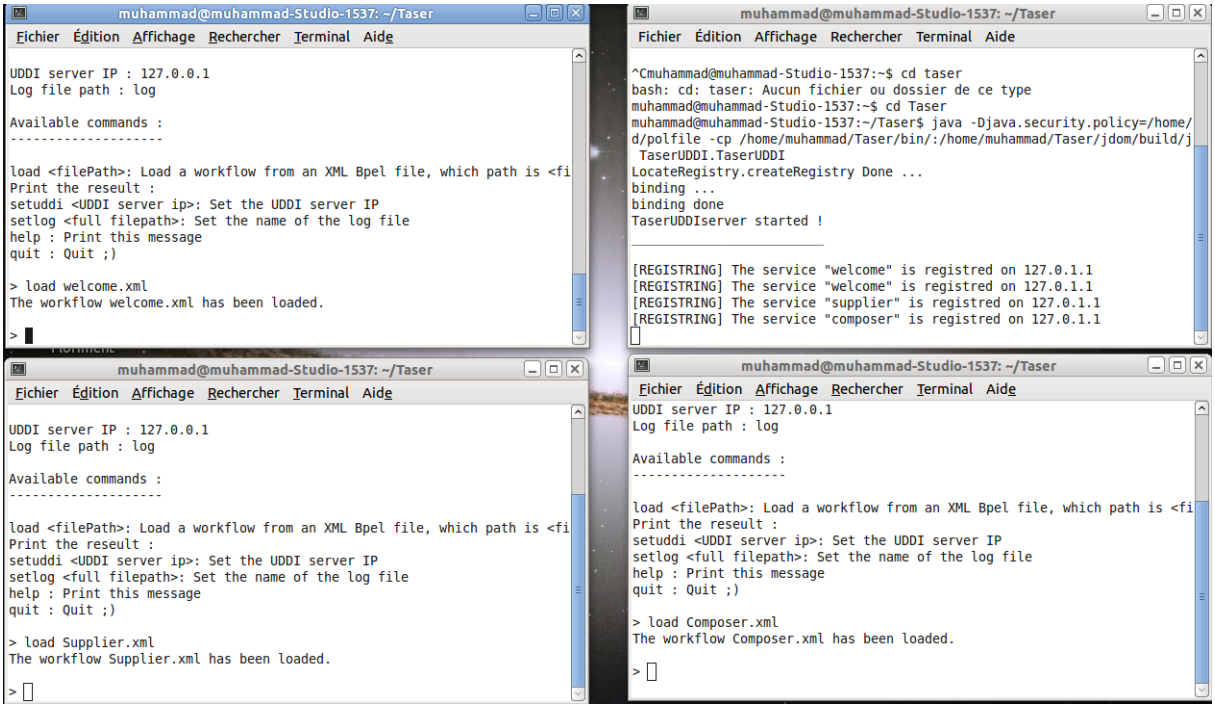

Figure 12: *Snapshot of Taser console mode after correction*

References

[1] http://www.w3.org.

[2] http://www.w3schools.com.

[3] oasis-open.org. August 2006.

[4] Anis Charfi, Tom Dinkelaker, and Mira Mezini. A Plug-in Architecture for Self-Adaptive Web Service Compositions. In *Proceedings of the 2009 IEEE International Conference on Web Services*, ICWS '09, pages 35–42, Washington, DC, USA, 2009. IEEE Computer Society.

[5] Onyeka Ezenwoye and S. Masoud Sadjadi. TRAP/BPEL: A framework for dynamic adaptation of composite services. In *Proceedings of the International Conference on Web Information Systems and Technologies WEBIST*, pages 1–6, 2007.

[6] Xavier Le Guillou. Une Approche Décenterlisée à base de chroniques pour la surveillance et le diagnostic de service web. Rennes, France, 2008.

[7] Xavier Le Guillou, Marie-Odile Cordier, Sophie Robin, and Laurence Rozé. Surveillance de compositions de web services. Deux approches distribuées à base de chroniques pour la surveillance et le diagnostic. volume 24, pages 189–225, 2010.

[8] Jeffrey O. Kephart and David M. Chess. The Vision of Autonomic Computing. volume 36, pages 41–50, Los Alamitos, CA, USA, 2003. IEEE Computer Society Press.

[9] Xavier Le Guillou, Marie-Odile Cordier, Sophie Robin, and Laurence Rozé. Chronicles for On-line Diagnosis of Distributed Systems. In *Proceeding of the ECAI 2008: 18th European Conference on Artificial Intelligence*, pages 194–198, Amsterdam, The Netherlands, The Netherlands, 2008. IOS Press.

[10] Xavier Le Guillou, Marie-Odile Cordier, Sophie Robin, and Laurence Rozé. Monitoring WS-CDL based Choreographies of Web Services. In *20th International Worshop On Principles of Diagnosis(DX'09)*, Stockholm, Sweden, 2009.

[11] Peyman Oreizy, Michael M. Gorlick, Richard N. Taylor, Dennis Heimbigner, Gregory Johnson, Nenad Medvidovic, Alex Quilici, David S. Rosenblum, and Alexander L. Wolf. An Architecture-Based Approach to Self-Adaptive Software. volume 14, pages 54–62, Piscataway, NJ, USA, May 1999. IEEE Educational Activities Department.

[12] Barbara Pernici and Anna Maria Rosati. Automatic Learning of Repair Strategies for Web Services. In *Proceedings of the Fifth European Conference on Web Services*, pages 119–128, Washington, DC, USA, 2007. IEEE Computer Society.

[13] Biplav Srivastava and Jana Koehler. Web Service Composition - Current Solutions and Open Problems. In *In: ICAPS 2003 Workshop on Planning for Web Services*, pages 28–35, 2003.

[14] Sattanathan Subramanian, Philippe Thiran, Nanjangud C. Narendra, Ghita Kouadri Mostefaoui, and Zakaria Maamar. On the Enhancement of BPEL Engines for Self-Healing Composite Web Services. In *Proceedings of the 2008 International Symposium on Applications and the Internet*, pages 33–39, Washington, DC, USA, 2008. IEEE Computer Society.